Safe, Fault Tolerant and Capture-Resilient Environmental Parameters Survey Using WSNs

Gianni Fenu and Gary Steri

University of Cagliari, Computer Science Department,
Via Ospedale, 72, 09124 Cagliari, Italy
fenu@unica.it, steri@sc.unica.it

Abstract. Sensor networks are one of the first examples of pervasive computing, which is characterized by the massive use of increasingly smaller and powerful devices. A cloud of sensors arranged in a given environment is in itself a great source of data; accessing this source in order to extract useful information is not a trivial problem. It requires correct sensor deployment within the environment and a protocol for data exchange. We also have to bear in mind the problem of data and sensors security: sensors are often installed in areas difficult to protect and monitor. In this paper we describe SensorTree, a functioning model and a simulator for a network of wireless sensors installed on the sea surface to measure parameters useful for determining the weather situation.

Keywords: Key management in wireless/mobile environments, Data retrieval, Data security, Hashchain, Network resilience.

1 Introduction

Recent progress in micro-electromechanical systems (MEMS), wireless communications and digital electronics has allowed the development of increasingly efficient and cheap multifunctional sensor nodes capable of communicating with other sensor nodes through low-range wireless technologies.

A Wireless Sensor Network (WSN) is a set of sensor nodes arranged within or very close to the phenomenon to be observed [1]. It can comprise different kinds of sensors (e.g. seismic, magnetic, infrared, acoustic, radar) useful for monitoring many environmental conditions [2].

Sensors are able to "collaborate" and preliminarily process data, sending not just coarse data to collector nodes [1]. Thanks to this characteristic, sensor networks are suitable for many kinds of applications, such as medical, military and security.

The creation of these networks can require some techniques originally developed for wireless ad hoc networks. However, the protocols and algorithms directly derived from ad hoc networks do not fulfill sensor network purposes. We can argue this point outlining the main differences between these two kinds of networks [3]:

- the number of nodes in a sensor network can be much greater than in an ad hoc network;
- sensor nodes are deployed with high density;

A.U. Schmidt and S. Lian (Eds.): MobiSec 2009, LNICST 17, pp. 180 – 189, 2009.

- sensor nodes are prone to frequent failures;
- sensor network topology can change frequently;
- sensor nodes mainly use broadcast communications, whereas ad hoc networks are based on point-to-point communications;
- sensor nodes have very restricted power, computing and storage capabilities.

The last point is one of the main constraints of sensor networks; sensor nodes are equipped with limited power supply units and, generally, it is not possible to recharge or change them. This paper is concerned with the design and implementation of a model for a real sensor network application, that allows for all the conditions described above and provides secure communication scheme and data transfer, by means of multiple hashchains and techniques for node protection against external attacks.

2 Model Architecture and Network Functioning

SensorTree model has been conceived to obtain weather conditions by monitoring temperature, wind speed and luminosity at the sea surface close to a beach or a coastal area. It uses a multi-hop communication scheme with sensors identified by a unique ID and one Sink node always active. Each node should be able to measure the three weather parameters and send the data to Sink, which should be able to determine the situation throughout the entire monitoring area.

Each node lies within a rigorous level architecture defined over the stretch of sea to be monitored. The upper bound of the area (landwards) is delimited by Sink node and the lower bound (seawards) is defined by the number (indefinite) of levels. So the total number of levels is given by the number of nodes and their transmission range, which determines the level's height and width.

The operating area of a SensorTree network can be represented as a series of overlapping rectangles, as shown in figure 1. Level dimension has been chosen so that

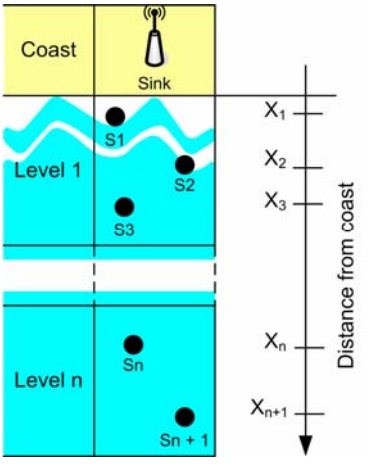

Fig. 1. Level architecture of SensorTree

transmission range of devices allows communication only between nodes at contiguous levels, in other words it is not possible to skip a level. The height of a level is a little more than half-transmission range and is equal to 5/4 of the width. Increasing level width risks compromising communication between a node of level n and a node of level n±1.

By virtue of this structure, each node resides only in one level (there are no overlapping areas) and has well defined coordinates within the area. We assume the devices are attached to a buoy and anchored to the sea bed, hence with negligible variations of their position.

2.1 Tree Construction

SensorTree network functioning is based on a tree structure which is built up along the levels of the monitoring area. The start-up of the entire survey system envisages a downward network growth starting from the Sink node; once Sink has been started, network construction begins turning on a sensor node (e.g. s1) in level 1. S1 notifies its presence sending a PowerOn request containing its position, battery level and other parameters[1].

Sink receives the request and sends to s1 a PowerOn reply communicating its identity, adding the node to its children list and marking it as "preferred" child. S1 immediately takes Sink as its root node (the only one for the first level).

Switching on another node (s2) in the same level, will produce nearly the same operation as above, but in addition:

- s1 does not reply to the PowerOn request sent by s2: requests sent by a node of the level n are handled only by nodes of levels n-1 and n+1;
- as soon as Sink receives s2 request, Sink adds s2 to its children list and checks its battery level: if it is higher than s1, then s2 will become the new preferred child.

The rules outlined above hold for any other node turned on in level 1; updating of the Sink's children list is very important.

Tree construction proceeds in the same way for the other levels, but from the second level onwards each node can choose between different roots. The choice of "preferred" root, as happens for children, is based on battery level, and variations therein can cause root changes.

2.2 Tree Maintenance and Convergence

The tree structure changes continuously; node battery level varies constantly, a few nodes can switch off or be destroyed and new nodes can join the network. So, a tree maintenance and convergence policy is required, that always guarantees the delivery of active nodes data to Sink.

Every T seconds (in simulations we chose T=10, but in similar contexts a longer time period can be used), each node sends to its neighbor nodes (children and roots, never nodes on its same level) a StatusRefresh notification containing battery level in addition to surveyed parameters. It is important to specify that T period starts a few

[1] In the simulation software, besides an address and the survey data, we used ports useful to different services and to test many instances of the application in the same machine.

instants after the node is switched on, so notifications are not synchronized. Obviously simultaneous notifications can occur but co-channel interference problems only arise for contiguous levels. The destination node (can be Sink too) receives the essential information indicating that its neighbor is still active and updates its lists using just received data, perhaps modifying preferred root and child.

Eventually some node fails; this means that it will no longer send StatusRefresh notifications. If a node stops sending send StatusRefresh notifications it will be deleted from its neighbor membership lists; if the node was a preferred node then preferred root and child must also be updated according to battery level of still active nodes. This mechanism has been implemented using a timer started at the same time as the last notification acceptance.

Should a node switch off or turn on before the timer has expired, probably no one would realize, except for the fact that when turned on again it sends a PowerOn request to detect neighbor nodes. In any case this request would be discarded (could be a duplicated ID) and the node would resume working as though nothing had happened.

2.3 Routing Tree

The tree construction described in the previous sections places much emphasis on node battery level and thus on estimating node life time. The choice of preferred roots and children is fundamental in determining routing paths used for data flow toward Sink.

From the above described rules it should be obvious that no node has a full view of the tree. Each node knows nodes on contiguous levels (its children and roots) but does not know what lies beyond. Only Sink can, at any time, perform a query on the tree and know all its details. Underlying nodes (not every one) can use the query to know next levels but they do not know anything about the previous ones.

In section 2.2 we stated that StatusRefresh notifications are not synchronized. This means that not every node has the same view of the tree at every instant. This problem is solved when it is required to know precisely the network status, that is when Sink requires a tree snapshot. This event triggers a data update of all sensors reachable from Sink, giving a full view of the monitoring area.

Tree tracking, like most of the operations performed on the tree, is recursive and is based on two simple steps:

1. Sink makes a StatusRefresh call to all its children which reply with a notification containing all data; Sink updates its children list and, if it is not empty, adds it to the procedure result;
2. The procedure is propagated recursively until the lower level (or until a tree interruption occurs) and the results returned to the upper level at each step.

This approach is not purely recursive because each root receives the first results simultaneously with propagation to next levels, not after reaching the end of the tree.

2.4 Nodes Querying

SensorTree simulator, in addition to tracking functionality, provides 4 modes to query the nodes. Queries do not require an immediate update of the lists: each node replies with data obtained during last refresh, such that no inconsistent data are sent. In fact, in these cases only one node is responsible for data sent to upper levels. Furthermore,

the data provided are at the most T seconds "old" and this can be neglected for T values within reasonable ranges. Anyway instantaneous data can be obtained performing a refresh, as previously described, before querying the tree.

All query modes are based on checking children lists owned by each root and on query propagation to preferred child. Also in these cases the recursive technique is adopted with the same specification described in the previous section.

The first query mode involves all nodes and is useful for obtaining data for all sensors contained in the network. The results can be used, for example, for calculating average values of parameters for the whole tree. The preferred root of each level is responsible for the data transmitted to Sink; this query is able to detect possible duplicated nodes on the network.

The query on one node, detected using its ID, returns all data surveyed from that sensor: temperature (°C), wind speed (m/s), luminosity (lx). Moreover, it returns the level where the node is situated, its coordinates and its battery level. The presence of a duplicated ID is not detected because the procedure stops at the first matching node; however, retrieved information can be used to perform a consistency check with initial node distribution.

Another mode is the level query, which returns all data of the nodes in the level specified as query parameter. It can be used to verify if a level is reachable and detect tree interruptions.

Finally, the query on a point allows to obtain survey data in a particular point of the monitoring area, specified by its coordinates. Obviously not necessarily does a node exist at the exact point specified, therefore the data obtained pertain to the sensor closest to that point. Unlike the other queries, propagation does not follow the preferred child but the child closest to the requested point; along with parameter values survey distance is also determined.

2.5 Automatic Rejoining

It clearly emerges from the above sections that tree construction cannot skip any levels: turning on a node in a level not contiguous to the lower one, will not change the tree structure because this node is too distant to be reached. However it is possible to continue tree construction because turning on other nodes in the underlying levels follows the same rules explained so far, so the first non reachable node is the root of an independent sub-tree. This scenario can be also obtained when every node of a level stop working: the preferred child of the previous level and the preferred root of the next one have no links in that level and cannot communicate.

This kind of situation is inconsistent because sub-tree data cannot reach Sink, but is not so compromising: turning on a node in the empty level suffices to join the two trees. The mechanism used is very simple and allows the functioning of one or more independent trees; while the level is empty, data arriving to the root of the lower sub-tree are stored in a buffer so, if the period of time is enough short, survey data won't be lost. When a node in the empty level is turned on, it sends a PowerOn request that is satisfied by the preferred nodes of the contiguous levels and the sub-tree is automatically rejoined. At this point, the level previously empty, works like other levels; the steps are shown in figure 2 (the bold line between two nodes shows preferred root and child):

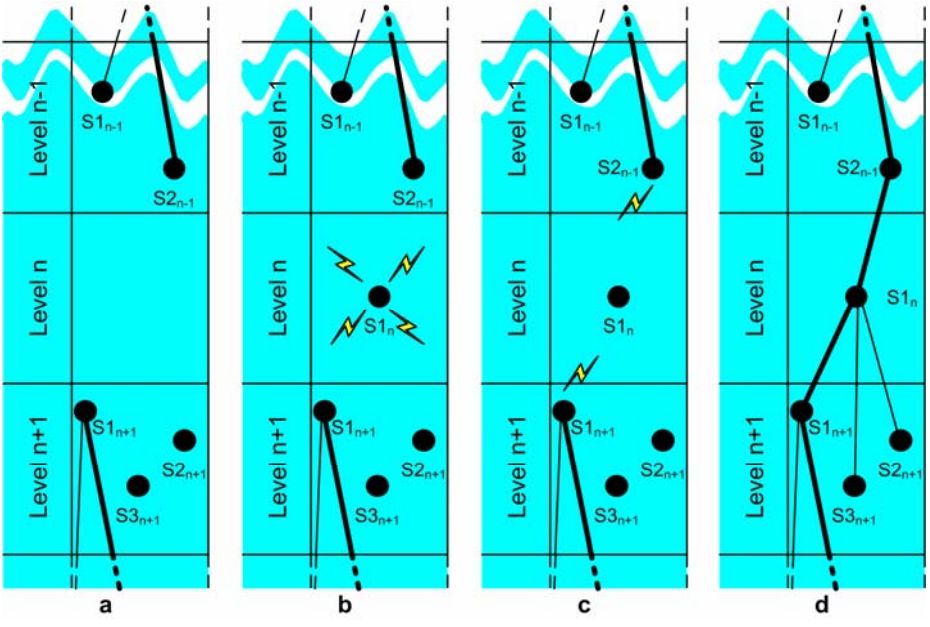

Fig. 2. Automatic rejoining steps. a) the level n is empty and there is an independent sub-tree starting from level n+1: $S1_{n+1}$ bufferizes survey data; b) $S1_n$ is turned on in the level n and sends a PowerOn request; c) $S2_{n-1}$ and $S1_{n+1}$ (respectively the preferred child and the preferred root of their levels) send a PowerOn reply; d) $S1_n$ is the preferred child of $S2_{n-1}$ and the preferred root of $S1_{n+1}$: the sub-tree is rejoined and data collected by $S1_{n+1}$ can reach Sink.

3 Data Integrity and Authenticity

Data flowing through the tree are not in themselves sensitive data and do not require specific policies to protect users' or third-party's privacy. However authenticity and integrity of data collected by Sink need to be guaranteed. For this reason, each message bound for Sink is accompanied by an authentication code computed by node's unique key, according to HMAC (keyed-hash message authentication code) standard:

$$H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text)) \quad (1)$$

In simulations we used SHA-1 algorithm (20 byte keys) and SHA-512 (64 bytes).

3.1 Nodes Initialization

Nodes need a key to authenticate communications directly addressed to Sink and another one to initialize a hashchain with its preferred root in order to send measured parameters. Since we do not use a public key infrastructure (PKI), prior to initializing communications with Sink we need to authenticate messages.

The initialization of a node is done during its installation, when it can load in main memory a key from a trusted source and use it to communicate with Sink (which

obviously knows keys associated to each node) without saving it in a secondary memory. But in this way, only the first initialization of the node can be done and only being able to physically reach the node. Subsequent activations may be necessary in the event of malfunction, temporary exclusion from the tree or after attack by extraneous nodes.

In order to obtain this functionality each node has to be able to execute a hash function (e.g. SHA-1) and a secure deletion algorithm (e.g. Gutmann with appropriate passes only, Schneier, DoD-3 or other simpler ones). Moreover, the node has to store a list of keys to be used for subsequent activations (number of possible activations depends on number of keys but, then again, the device's life is limited by its battery duration). Memories considered here take into account dimensional specifications of the sensor network.

After the first initialization, a node that has to be remotely re-activated expects to receive from Sink the hash for the first key on the list (only Sink can compute it). As soon as this is received, the node computes the hash for the key and matches it with the one received, verifying that activation was actually started by Sink. At this point, the node loads the second key of the list in the main memory deleting it in secure mode from flash memory. The first key (which remained in main memory since first initialization) is now active and can be used to authenticate the messages directed to Sink. The activation sequence is shown in figure 3.

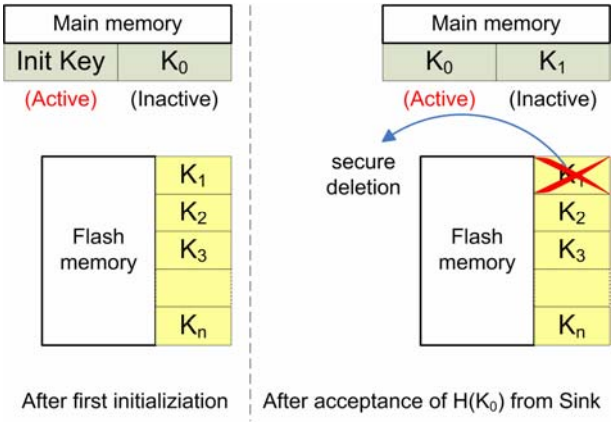

Fig. 3. Keys activation sequence

3.2 Data Collection and Hashchains

Data collection passes at each level through preferred root of the nodes, which is responsible for data sent to Sink. Obviously, in the first level, each node can send the data directly to Sink, which can therefore be authenticated using HMAC standard. Otherwise, each node can start a hashchain with Sink to guarantee data authenticity and integrity. This method is used for next levels of the tree, where each preferred root plays the role of Sink, in order to avoid individual node data being sent directly to Sink for authentication and the resulting generation of excessive traffic volumes.

In order to initialize the chain, each node, possible root, stores the hashes of a set of initialization keys owned by underlying nodes. The pool of keys is loaded in main memory as explained in the previous section for activation keys, while the hashes of the keys can be stored in the flash memory.

The chain is started by sending the key's hash to the preferred root together with first data hash; the next block contains first data, the hash of the second and so on. The chain can be interrupted and subsequent reinitializations cannot be done using the same hash. To solve this problem each root can store the hashes of different keys for each node, otherwise can expect to receive the nth hash of the key at first initialization, the n-1th at the second and so on, creating a one-time password authentication scheme. Root node storage and hashchain are shown in figure 4:

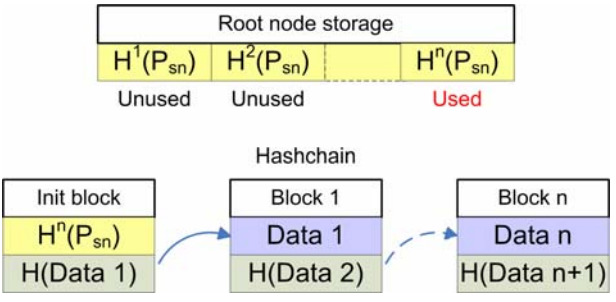

Fig. 4. Hashchains and one-time passwords

Using this mechanism a potential attacker wanting to enter the chain should be able to retrieve measurement data of next block using the hash of current block. Observing enough transmissions, an attacker will notice that some measurement data recur in the chain and analyzing the hashes can try to guess the content of the next block. To avoid this kind of attack, means ensuring that identical survey data generate different hashes, introducing additional data in the computation process. Introducing values obtained by a pseudo-random number generator (PRNG) can be useful but this does not significantly enhance security; inserting values obtained by a true-random number generator (TRNG) is the best solution. In simulations, the true-random values has been generated using parameter measurement intervals: the expiry time between one measurement and another is not extremely rigid and is influenced by hardware delays due to measurement devices and clock. The delays and the total time from last measurement are measured with an accuracy higher than interruptions (e.g. milliseconds or nanoseconds) and added to data packet (which also contains unique IDs for each node). The probability that an attacker guesses the exact instant of data measurement and sending is really low.

Once the preferred root has collected data from each child and verified their authenticity, it sends them directly to Sink using a JumboData packet that is forwarded on the routing tree path. This packet does not contain authentication data of individual nodes but only an authentication code computed by the root according to HMAC standard.

4 Capture-Resilience

The most difficult problem to solve in this kind of application is the physical protection of the nodes. An attacker can steal them, copy their content, clone them or exchange them with other devices. In order to protect the nodes we need to ensure that even if the attacker does steal the node, he is unable to extract useful information for accessing the network.

In the nodes activation procedure and for using the keys for measurement data sending, we have chosen to store the active key and the next key of the activation pool in the main memory (RAM) alone, performing a secure deletion from the secondary memory (flash memory) of the loaded keys. The latter step is very important because, even after a flash memory erase operation, the transistors do not return fully to their initial state, thereby allowing the attacker to distinguish between previously programmed and not programmed transistors, and thus restore information from erased memory [4].

We also have to prevent the attacker from accessing the main memory content, for example by creating tamper-protection for the node which forces node shutdown in the event of intrusion. But shutdown in itself is not sufficient, because RAM memories are also prone to data retention problems: using a key for a long period of time can cause long-term retention effects in the memory [5], leaving useful traces for restoring information. Therefore memory cells should not store data for too long a period of time (e.g. performing a bit flipping) and in the event of tamper detection it is necessary to apply a reverse current which reverses the electromigration stress due to long-term retention [5]. In this way node theft and access to its content render the node unusable for joining the network.

In our tests and simulations we verified the functioning of solutions proposed in previous sections proving that latency times of joining procedures and query propagations allow a correct data delivery to Sink node; for this purpose we also considered latencies introduced by capture-resilience measures for keys activation sequences and nodes initialization.

5 Conclusion

The model described above has allowed to develop a simulator which fulfils all requirements for the correct functioning of SensorTree network. The proposed model is also the cornerstone for an ongoing development project for testing and creating the network.

The use of the tree structure enables efficient communication between the nodes minimizing the paths to Sink. In spite of this advantage, the tree structure introduces geometrical constraints on level dimensions;: However, this problem can be solved by introducing intra-level communications and designating some groups of nodes as landmarks. Intra-level communication can also be useful for differentiating data collection modes and for simplifying pure recursive operations on the tree.

Level subdivision enables critical network areas to be identified and to act simply on those areas, maintaining a regular functioning of the other levels (even with independent trees). Even simply adding one node to the network can create a critical

situation when the number of nodes is very large. This is the reason why it may be necessary to develop a control protocol to handle this situation, to quickly solve tree interruptions and to allow Sink node mobility.

The choice to rely on hash algorithms and shared keys for message authentication and integrity well matches sensor node capabilities, whose most recent commercial versions have barely sufficient resources to operate on asymmetric key encryption algorithms [6]. This choice has also made it possible to reduce data overheads introduced by encryption algorithms, especially for very large data blocks. However, the use of cryptography (even if symmetric) could be useful for ensuring greater protection of some communications, for simplifying key redistribution and for ensuring data privacy when necessary.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer Networks 38, 393–442 (2002)
2. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next century challenges: scalable coordination in sensor networks. In: ACM MobiCom 1999, Washington, pp. 263–270 (1999)
3. Perkins, C.: Ad Hoc Networks. Addison-Wesley, Reading (2000)
4. Skorobogatov, S.: Data Remanence in Flash Memory Devices. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 339–353. Springer, Heidelberg (2005)
5. Gutmann, P.: Data Remanence in Semiconductor Devices. In: 10th USENIX Security Symposium Proceedings, Washington (2001)
6. Tan, H., Jha, S., Hostry, D., Zick, J., Sivaraman, V.: Secure Multi-hop Network Programming With Multiple One-way Key Chains. In: WiSec 2008, Alexandria (2008)
7. Çamtepe, S.A., Yener, B.: Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks. IEEE/ACM Transactions on Networking 15(2) (2007)
8. Akyildiz, I.F., Vuran, M.C., Akan, O.B.: A Cross-Layer Protocol for Wireless Sensor Networks. In: Conference on Information Science and Systems (CISS 2006), Princeton (2006)
9. Gutmann, P.: Secure Deletion of Data from Magnetic and Solid-State Memory. In: 6th USENIX Security Symposium Proceedings, San José (1996)
10. Giannotti, F., Pedreschi, D.: Mobility, Data Mining and Privacy Geographic Knowledge Discovery. Springer, Heidelberg (2008)
11. Tracy, L.T., Roy, S.: A reservation MAC protocol for ad-hoc underwater acoustic sensor networks. In: WUWNet 2008, San Francisco (2008)
12. Badia, L., Mastrogiovanni, M., Petrioli, C., Stefanakos, S., Zorzi, M.: An optimization framework for joint sensor deployment, link scheduling and routing in underwater sensor networks. In: WUWNet 2006, Los Angeles (2006)